

# Auto Increment

- First you need to recognize that (like assignment) autoincrement must be implemented at both the statement and expression levels
  - Statement level `a++`;
  - Assignment level `b = a++`;
- You will handle this very similar to the way you handled assignment. Make sure that the statement clears the stack and the expression leaves a value on the stack

# Pre and Post

- You will also have to handle pre and post increment.
- At the statement level, they both do the same thing, so `a++;` is the same as `++a;`
- At the expression level, you want the following to occur:
  - `a = 9; b = a++;`
  - Causes b to be set to 9 and a is set to 19
  - `a = 9; b = ++a;`
  - Causes both a and b to be set to 10.

# Implementation of `b = ++a;`

- Refer to my directory  
`/user/faculty/hilzer/yacc_mod`
- Notice that `++` is recognized in `lex.l` as `PP` and is treated like it is an integer with the constant value 1. That will be used later because autoincrement requires that we increment by 1

# Autoincrement Productions

- Insert the following productions in c.y

stmt: incr1

expr: incr2

incr1: PP ID  
      | ID PP

incr2: PP ID  
      | ID PP

# Semantic Action

- Here is the semantic action for
- incr2: PP ID {  
code3(varpush, (Inst)\$2, eval); /\* push var on stack \*/  
fprintf(ap, "%d %s", (progp-progbase-3), "varpush\n");  
fprintf(ap, "%d %s\n", (progp-progbase-2), \$2->name);  
fprintf(ap, "%d %s", (progp-progbase-1), "eval\n");  
code3(constpush, (Inst)\$1, add); /\* increment \*/  
fprintf(ap, "%d %s", (progp-progbase-3), "constpush\n");  
fprintf(ap, "%d %lf\n", (progp-progbase-2), \$1->u.val);  
fprintf(ap, "%d %s", (progp-progbase-1), "add\n");  
code3(varpush, (Inst)\$2, assign); /\* leave a value on stack \*/  
}